Grsecurity: Advanced Linux Hardening

James Buszard-Welcher

jwelcher@lbl.gov

UNIX Systems Engineer
Information Technologies & Services Division

June 6th, 2003

Overview

The Gist

We'll explore hardening the Linux kernel using grsecurity, separating root from omnipotence

The Goal

Get you interested, give you enough information to start to play with it.

The Audience

- **Linux/Unix admin experience**
- **Basic familiarity with security issues**

The Source of Insecurity

- Software has bugs which cause unexpected things to happen
- Misconfigured or badly designed software can also result in unpredictable behavior
- These bugs can be used to attack a system
- Increasingly these attacks are packaged as rootkits

Basic Security

(not a complete list)

- Minimal system install, careful filesystem layout
- Current updates/patches, sit on mailing lists
- Turn off all unneeded services, TCP wrap them
- Tripwire, or other integrity checker
- Much logging, monitoring
- Host firewall
- No unencrypted access, perhaps safe-word cards
- etc.

Basic Security is often insufficient

How far can you trust:

- Your users?
- Any program that one of your users might have uploaded?
- Any program one of your users might write?
- Any network service (I.e. web CGI) they might run?
- Any machine they log in from? (think Linux keystroke logger… think VNC or back orifice or any of a bunch of Windows back doors…)

ROOTKITS are designed to make anyone a hacker.

Any Solutions?

- Chroot?
- Other restricted shells?
- Run OpenBSD instead?

Anything else?

Kernel Hardening Philosophy

- Assume that users are hostile.
- Assume they will try to get root.
- Try to prevent them from getting root by removing common paths. Make them work.
- Minimize their damage by reducing their capabilities even if they get root.

Kernel Hardening Systems

GRSecurity

LIDS: “Linux Intrustion Detection System”

Openwall

Other Unix:

*BSD: Secure Kernel Levels

Trusted Solaris

Grsecurity Features

- Intrustion Detection Auditing
- Disable traditional attack vectors such as stack smashing via PaX
- Remove easy to guess system results via OpenBSD-based randomization of RPC XIDs, privileged ports, PIDs, IP Ids, etc.
- Hardening syscalls: chroot(2), ptrace(2), mmap(2), etc.
- Trusted Path Execution
- Process-based ACLs (Access Control Lists)

Detect via Extra Auditing

- Exec (with arguments)
- Chdir(2)
- Mount(2)/unmount(2)
- IPC (semaphore, message queue, shared memory) creation/deletion
- Certain signals: SIGSEGV, SIGABRT, SIGBUS
- Failed forks
- Ptrace

etc.

PaX

- Included with grsecurity, but it's own project.
- Different types of memory protection
- Implements non-executable VM pages on architetures that don't support the non-executeable bit (32-bit Intel)
- Address space layout randomization for ELF binaries (ASLR)
- Prevents stack smashing and heap overflow attacks

Trusted Path Execution (TPE)

Keeps users from executing untrusted binaries (those not in root-owned non-world writable directories)

Hardened against evasion

- Silent removal of glibc environment variables that allow arbitrary code execution (eg. LD_PRELOAD)
- TPE checks implemented in mmap(2) (stops /lib/ld.so <executable> evasion)

Grsecurity “Low” Additions

linking restrictions

fifo restrictions

random pids

enforcing nproc on execve()

restricted dmesg

random ip ids

enforced chdir("/") on chroot

Grsecurity “Med” Additions

- random tcp source ports
- altered ping ids
- failed fork logging
- time change logging
- signal logging
- deny mounts in chroot
- deny double chrooting
- deny sysctl writes in chroot
- deny mknod in chroot
- deny access to abstract AF_UNIX sockets out of chroot
- deny pivot_root in chroot
- denied writes of /dev/kmem, /dev/mem, and /dev/port
- /proc restrictions with special gid set to 10 (usually wheel)
- address space layout randomization

Grsecurity “High” Additions

- additional /proc restrictions
- chmod restrictions in chroot
- no signals, ptrace, or viewing processes outside of chroot
- capability restrictions in chroot
- deny fchdir out of chroot
- priority restrictions in chroot
- segmentation-based implementation of PaX
- mprotect restrictions
- removal of /proc//[mapslmem]
- kernel stack randomization
- mount/unmount/remount logging
- kernel symbol hiding

Controling Grsecurity

- What options you compile in
- Runtime sysctl controls
- Control of ACLs via /etc/grsec/acls
- gradm program acts as interface

Enabling / Disabling

Enable Grsecurity

```
gradm -E
```

Become UberUser (this shell is free)

```
gradm -a (*)
```

Disable Grsecurity ACLs from system

```
gradm -D (*)
```

(*) requires UberPassword

gradm -h for help

ACL Overview

- Process-based
- Enable/disable/admin modes
- Hidden and Protected processes
- Read, Write, Append, Execute, Hide modes for file objects
- Inheritance

ACL Structure

```
<path of subject process> <optional subject modes> {
        <file object> <optional object modes>
        [+|-]<capability>
        <resource name> <soft limit> <hard limit>
        connect {
           <ip>/<netmask>:<low port>-<high port> <type> <proto>
        }
        bind {
           <ip>/<netmask>:<low port>-<high port> <type> <proto>
        }
}
```

Postfix ACL p1

```
/usr/sbin/postfix o {
        /var/spool/postfix rw
        /var/spool/postfix/lib rx
        /var/spool/mail w
        /var/mail w
        /dev/log rw
        /dev/null rw
        /dev/urandom r
        /etc/aliases
        /etc/mailman.aliases rw
        /etc/postfix rw
        /etc/postfix/postfix-script rx
        /etc r
        /etc/grsec h
        /lib rx
        /usr/lib rx
        /usr/share/zoneinfo r
        /var/tmp
        /  h
```

Postfix ACL p2

```
            -CAP_ALL
            +CAP_DAC_OVERRIDE
            +CAP_KILL
            +CAP_SETGID
            +CAP_SETUID
            +CAP_SYS_CHROOT

            connect {
                    0.0.0.0/0:53 stream dgram ip tcp udp
                    0.0.0.0/0:25 stream ip tcp
            }

            bind {
                    disabled
            }
    }
```

ACL Subject Modes

```
h This process is hidden and only viewable by processes with the v mode.
v This process can view hidden processes.
p This process is protected; it can only be killed by processes with the k mode, or by processes
   within the same subject.
k This process can kill protected processes.
l Enables learning mode for this process.
d Protect the /proc/<pid>/fd and /proc/<pid>/mem entries for processes in this subject.
b Enable process accounting for processes in this subject.
P DISABLES the PAGEEXEC feature of PaX on this subject.
S DISABLES the SEGMEXEC feature of PaX on this subject.
M DISABLES the MPROTECT feature of PaX on this subject.
R DISABLES the RANDMMAP feature of PaX on this subject.
G ENABLES the EMUTRAMP feature of PaX on this subject.
X ENABLES the RANDEXEC feature of PaX on this subject.
O Override the additional mmap() and ptrace() restrictions for this subject.
A Protect the shared memory of this subject. No other processes but processes contained
 within this subject may access the shared memory of this subject.
K When processes belonging to this subject generate an alert, kill the process
C When processes belonging to this subject generate an alert, kill the process and all processes
   belonging to the IP of the attacker (if there was an IP attached to the proce ss)
T Ensures this process can never execute any trojaned code
o Override ACL inheritance for this process.
```

ACL Object Modes

```
r This object can be opened for reading.
w This object can be opened for writing or appending.
x This object can be executed (or mmap'd with PROT_EXEC into a task).
a This object can be opened for appending.
h This object is hidden.
t This object can be ptraced, but cannot modify the running task. This
   is referred to as a 'read-only ptrace'.
s Logs will be suppressed for denied access to this object.
i This mode only applies to binaries. When the object is executed, it
   inherits the ACL of the subject in which it was contained.
R Audit successful reads to this object.
W Audit successful writes to this object.
X Audit successful execs to this object.
A Audit successful appends to this object.
F Audit successful finds of this object.
I Audit successful ACL inherits of this object.
```

Resources

RES_CPU	CPU time in millisecond
RES_LOCKS	Maximum file locks
RES_CRASH	Number of times to allow process crashes
RES_FSIZE	Maximum file size in bytes
RES_DATA	Maximum data size in bytes
RES_RSS	Maximum resident set size
RES_NOFILE	Maximum number of open files
RES_MEMLOCK	Maximum locked-in-memory in bytes
RES_STACK	Maximum stack size in bytes
RES_AS	Address space limit in bytes
RES_NPROC	Maximum number of processes

Process Capabilities

- /usr/include/linux/capability.h

Each process has an associated capability defined above which governs whether the kernel allows certain actions.

Grsecurity only allows certain processes to have certain capabilities as defined in /etc/grsec/acl, regardless of UID.

Utilizing grsecurity

- /etc/grsec/acl
- gradm
- UBERroot password
- sysctl

The Learning Mode

Give a process unlimited power, but take notes!

- Add entry for that process to /etc/grsec/acl where that process is put into learning mode and ALL capabilites are removed (draw a line in the sand)
- Reload the acl and then run that command
- Any “rule violations” are logged to syslog
- Extract a better ACL than you could write from the log via:

  gradm -L /var/log/messages -O /tmp/learned

5. Incorporate /tmp/learned as you see fit into acl

Examples

How does Trusted Execution Path work?

Make it ping!

With the default rules, ping won’t work for a normal user. Let’s find out why and fix it.

Gotchas

- I can’t ping
- X-forwading doesn’t work
- Certain network delays
- Everything is broken!!

TODO

- Generally make "production" ready (resolve gotchas)
- Workable Desktop config (X/Gnome, etc. work as expected)
- Performance Checks
- "Canned" ACL for common LBL applications
- Indicative tests
- Test out some rootkits on it

References

- http://www.grsecurity.net
- http://pageexec.virtualave.net/docs/pax.txt
- http://www.lids.org/lids-howto/